NATIONAL OS9 NEWSLETTER

Editor : Gordon Bentzen
8 Odin Street,
SUNNYBANK Qld. 4109.
(07) 345-5141

JANUARY FEBRUARY 1989

NATIONAL OS9 USER GROUP NEWSLETTER

EDITOR : Gordon Bentzen

HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 Users Group.

Well here we are in the year of 1989 and already one month has gone. We do hope to bring you something special in this our January / February edition. Firstly however, an apology to some members. The editorial of last Dec. advised that we intended to mail out back issues of last year's newsletters to bring all current memberships to a common expiry date, renewals being due September 1st '89. Well we did do this, in fact I suspect that some members would have finished up with at least one duplicate copy. We thought it better to err on the side of too many rather than have someone miss out. Now, at last, comes the apology - The back issues mailed did NOT include the July '88 or August '88 newsletters as we stated in Dec. Sorry about that. The task of reprinting back copies turned out to be larger than expected and I (Gordon) just slipped up. If you would really like a copy of Jul, Aug please drop me a line, we have Eight only JUL'88 and Three only AUG'88 as spare copies, first in best dressed.

PUBLIC DOMAIN SOFTWARE

We do have some public domain software which is available to members of the National OS9 User Group on the following basis.

[1] Mail us OS9 formatted disks addressed to 'National OS9 User Group' c/o the address shown on our cover.
[2] PLEASE do include postage stamps with disks for return postage (value the same as stamps to mail to us)
[3] Include Two Dollars ($2.00) for each disk. This is a copy charge per disk (any tracks 35,40,80 SS or DS)

This copy charge was established to cover any costs to us in obtaining public domain material, assist in the purchase of such material from the U.S. OS9 Group and to allow us to re-reimburse our contributors for their costs in sending us their material.

I do have a few disks which have been mailed in by members, either without return postage or without the $2 copy charge, or even without either. If you have mailed a disk or two and not included the above, then your request has not been fulfilled, and won't be until you do send same. Sorry but rules are rules.

Now, having got that bit off my chest I can move on to what is currently in the Public Domain library.
We have the U.S. OS9 User Group Disks from disk0 to disk42 , except 27,28,41. We have these stored on 80 track double sided disks. These are arranged in 'Volumes' 1 thru 9 with each volume containing four or more of the original issue disks (35tr). All programmes and listings which appear in the newsletter are also included in the library. Now, the most difficult part is to find a means to give you sufficient detail to enable each of you to select just what might be useful to you. For example, a hierarchial directory print-out of U.S. disks runs to 21 printed pages. At best, these file names on their own leave a lot to the imagination, so one of our members, John Usher, has gone to the trouble of merging all the 'Help' files from these disks. When listed to the screen or printer this file makes a good reference of all the U.S. disks, as well as just what each file or programme is all about. Well, this is just what we need, why not just list this file in the newsletter ? 126 pages that's why. Yes when listed to the printer and formatted for the normal 8 inch by 11 inch 80 col. page, you will finish up with 126 printed pages. John has also made an alphabetical index of all files which runs to a further five pages.
Well do you have any suggestions on this subject ? The only thing we can come up with is to ask you to send us a disk (rules above apply) and we can give you a copy of index and help files.

Once you are able to decide on which disks or volumes that you would like, just let us know. Remember that you can have a volume for the same copy charge as a disk and get at least four times the material. By far the easiest for us is a simple backup of a volume, 80 track format.

Oh yes ! almost forgot, have you seen ShellPlus ? This version of 'Shell' replaces the original shell in Tandy's OS9 Level 2 and adds quite a few nice features. One that we like is that the OS9 prompt can include the current

window, like :- OS9[W2]: or OS9[TERM]: This is in our library and is essentially two files, one the executable file and the other a doc file which lists to about four pages of documentation.

IS IT TRUE ?? Well we have also heard the rumours that Tandy will drop the CoCo here in Australia and that this will follow the lead of their parent in the U.S.

You will have already seen some CoCo software at clearance prices, and we understand that some hardware items for the CoCo are no longer available from the Mt. Druitt warehouse. So if you missed out on something from your Christmas wish list, it may be wise it go looking now.

This will not help OS9 because the CoCo still does provide a reasonably priced medium for the Fantastic OS9 operating system, and I for one expect my CoCo and OS9 to continue for some time yet.

We are aware of course that OS9 runs on other 6809 and 68K machines and in fact some members of our National group do use machines that were not made by Tandy. We are now more interested than ever in these systems as we will need to find an alternative in the long term. Come on, how about it, let us share some of the secrets of your favourite system using OS9.

THIS EDITION :-

Making the most of 'C' Part 2 by --Rosko!--
This part 2 is an excellent follow-on from the very welcome article presented in the December newsletter. Many thanks for your support Rosko.

OS9 For Beginners Part 3 by Bob Devries
We hope that this will help those not yet familiar with OS9

Running a Terminal under OS9 Part 2 by Don Berrie

TempConvert by Bob Devries
This is a simple fahrenheit to celcius conversion programme written in 'C'

NEXT MONTH We have the word that Bob Devries will be presenting a database series written in 'C' which will develop to a very usable database.

oooooooooo0000000000oooooooooo

Making the Most of Microware C - Part II

by --Rosko!--

NOTE: requires some knowledge of C and assembler (6809 or 68000)

Apparently, some people liked Part I so I seem to be writing Part II now. Some of you may even be starting to write some good little handy functions, so it's probably a good time to discuss what to do with a few hundred "good little handy functions"! You may also like to know a little more about those taboo's I mentioned last time... so read on!

When you compile or assemble anything with the C Compiler's assembler (i.e. c.asm, rma, or r68) you don't just get raw machine language you can run. The assembler generates what is called a Relocatable Object Format (ROF) module. This module has some overhead detailing the number of static (permanent) variables declared, the global variables and functions defined, and all the references to variables and functions outside the file. This allows you to write code in separate files, assemble them, and link the ROF modules together into the one executable file. It forms the basis behind library files and compiling separate C files into one program.

Because ROF modules involve this overhead, it's accepted as a good idea to group related functions into one ROF module, so less disk space is used and compilations are "a little quicker". To get more than one function in one ROF module, you keep the functions you think are related in one source file. Going back to the example of the standard string functions, strlen(), strcmp(), strcpy() and strcat() are all in the one source file so when they were compiled they all turned up in the one ROF module, strings_c.

When you write a C program which uses strings, you will more than likely use at least two of these functions and maybe all four, so you aren't wasting memory including unused code, and your library files are smaller. You can probably guess that if you threw everything into one ROF module, then you would be wasting memory by including lots of functions your program will never use. When you start grouping functions in one file, you have to start thinking about how often you will call these functions from one program.

Having grouped your functions into ROF modules, you can then make what is called a Library file. Basically, this is just a group of ROF modules merged into one file, in a certain order. When you write a program which calls one of your functions from the ROF modules, you can tell the linker (rlink, l68 or c.link) to search through the library file and extract the ROF module containing your function to add to your program. This is the basis of the C compiler's standard library.

Notice that I mentioned order in the above paragraph. In library files, it can be important to arrange your ROF modules in a certain order. This is because when the linker searches through the library file, and extracts an ROF module for your program, it continues to search for ROF modules with any of the functions required by the module it extracted. If, say, the strings_c module appeared before the printf_c module, and printf() called strlen(), the linker would first skip past strings_c to find printf(), then start looking for strlen(). But it has already passed strlen() in strings_c... so you can see that strings_c must come <u>after</u> printf_c.

All this is quite confusing, so an example is probably in order. You might want to write a package of financial calculations like loan estimate and compound interest functions, and perhaps some statistical functions like averages and standard deviation. You would group some into one file because they are likely to be called together, such as the statistical functions, and some separately as they wouldn't be used too regularly. When all your functions are compiled into ROF modules, you would merge them all into a Library file. Some of your functions might call other functions, so their ROF modules should be put at the top of the library. When you work out a sure win method for the stock market and write the ultimate program, you could compile it by typing:

```
cc makemillion.c -l=/dd/lib/finance.l
```

and the linker would go looking for your functions in the library file finance.l in your LIB directory.

It is interesting to note that you can write some source files in assembly language and some in C, and still produce ROF modules that can be put into a library file. The code in a ROF module is machine code, so it doesn't matter what it was written in!

Well, that's getting boring so let's discuss the reserved registers taboo.
6809 C reserves U, Y, DP and S; 68000/68020 C reserves D4-D7 and A2-A7. This doesn't mean you <u>can't</u> use them, just that C already uses them. If <u>you</u> want to use them you should understand what they are used <u>for</u>, and ensure they are restored to how C had them if you change them.

Firstly, whether you write in C or assembly language, your code uses a system stack to keep track of where a function was called from, so it can go back there when it does an RTS. C functions use the stack to pass arguments also. In 6809 C the stack is register S; in 68000/68020 C the stack is register A7, designated SP. DON'T PLAY WITH IT!

Any global or static variables you declare are kept at the front of your program's memory pool, and an index register points to the start. 6809 C uses Y, 68000/68020 C uses A6. You can use this register to access global variables from your assembly language functions by indexing from the register. For example, to access the global integer <u>errno</u> you could:

```
LDD errno,Y             or          MOVE.L errno(A6),D0
```

6809 C also supports direct page variables. These are in the first 256 bytes of the global and static area. If you know that a global variable is declared as direct page in C, you can access it from assembly language like:

```
LDD <freddo
```

68000/68020 C reserves the A5 register. I have noticed that in the latest version of the 68020 C compiler, A5 is linked at the start of each function, and unlinked before RTS. I don't know why... it doesn't appear to be used for anything yet. Until I can find out why, I suggest you LEAVE IT ALONE!

The remainder of the registers (i.e. U for 6809, D4-D7/A2-A4 for 68000/68020) are used by the compiler for any variables declared as register variables. As you can guess, 6809 C is limited to one register variable per function, whereas 68000/68020 C is "limited" to four data and three pointer register variables per function.

If you want to use any of these registers in your assembly language functions, you should first push them on the stack:

```
PSHS U,Y            or          MOVE.L D2-D7/A2-A6,-(A7)
```

and reclaim them before RTS'ing:

```
PULS U,Y            or          MOVE.L (A7)+,D2-D7/A2-A6
```

For 6809, if you would use RTS right afterwards, you can save code with:

```
PULS U,Y,PC
```

If your function calls another which may require access to global variables, you should either restore the global/static pointer (Y or A6) before BSR, or simply not use Y or A6.

To finish this month (I'm sick of typing again), here is a little function which allocates string space for a temporary string, so it can be copied before it disappears. Just pass a pointer to where your string is now, and it will either pass back a pointer to the new string or NULL which means no memory. You can discard the string when you are finished with it by passing the pointer to the string to free(). READ THE COMMENTS as they help explain the code.

```
* strsave.a - 6809 code for Microware's 6809 C compiler
*             allocate memory for a string, and save it
*             string may be deallocated using free()
* --Rosko!-- 15/5/88
* 14/12/88 - noticed it still used inoptimal strlen algorithm... fixed
*
        psect strsave_a,0,0,0,0,0

* char *strsave (s) char *s;
strsave:
        ldx   2,s
        bne   ss10    if pointer is NULL, don't go any farther
        clra
        clrb
        rts

ss10    tst   ,x+     look for NULL
        bne   ss10    on passing this point, X = end of string
        tfr   x,d     get end address in D
        subd  2,s     subtract start address
```

```
        pshs  d
        lbsr  malloc  ask for that many bytes
        leas  2,s     we pushed 2 bytes, so take them off again
        cmpd  #0      if zero, no memory so return zero
        bne   ss20
        rts

ss20    ldx   2,s     get start of string again
        pshs  u,d     want to use U, so must save it
        tfr   d,u
ss30    lda   ,x+     copy from temp. into new string
        sta   ,u+
        bne   ss30
        puls  u,d,pc  used U, so restore it; reclaim pointer also

        endsect

* strsave.a - save a string into external memory.
*            memory can be returned to system using free()
* --Rosko!-- 8/7/88 68000 version
*
        psect   strsave_a,0,0,0,0,0

* char *strsave (s) char *s;
strsave: tst.l   d0       is there something to save?
        beq.s   strsave3 don't bother if null string
        movea.l d0,a0    now, get length of string
        move.l  d0,-(sp) save pointer for later

strsave1 tst.b   (a0)+
        bne.s   strsave1 on passing this point, a0 = end of string
        sub.l   a0,d0    subtract from start of string
        neg.l   d0       gives length of string inc. null

        bsr     malloc   request memory from malloc (# bytes in D0)
        movea.l (sp)+,a0 reclaim pointer to string
        tst.l   d0
        beq.s   strsave3 don't bother if no memory
        movea.l d0,a1    get pointer to memory
strsave2 move.b  (a0)+,(a1)+
        bne.s   strsave2 copy from temp. string into new
        rts

strsave3 moveq.l #0,d0
        rts

        ends

C U NeXT time()
```

--Rosko!--

oooooooooo0000000000oooooooooo

SETIME PATCH :-

Here is a patch for the setime module which has kindly been sent to us by Robin Oosterbeek. I have used your patch Robin and it works well. O.K. over to you now Robin :-

I, like many others have become sick of the American way of entering the date, therefore I set out to make it the Australian way, so I modified "setime" to make it DD/MM/YY HH:MM:SS instead of YY/MM/DD HH:MM:SS.

```
modpatch -s
l setime        link setime
c 66 79 64      change from yy to dd
c 67 79 64
c 6C 64 79      change from dd to yy
c 6D 64 79
c A8 E4 62
c B0 62 E4
v               validate   (verify)
<eof>
```

I hope that these few changes are helpful for all - Robin Oosterbeek.

oooooooooOOOOOOOOOoooooooooo

STARTING OUT WITH OS9

Modifying your device descriptors.

One of the things everyone seems to want to do when they start to use OS9, is to modify the disk drive descriptors so that they match the drives that they have. The most important of these is the stepping speed of the head from track-to-track. That item, along with the number of tracks etc is found in the device descriptors, one for each drive you have in your system. The descriptors are D0, D1, D2 and D3. All four are used if you have four single-sided drives, and less if you have fewer or double-sided drives. The Colour Computer can only access three double-sided drives, but these can be of varying sizes. For instance, you could have one double-sided and two single-sided drive (NOT recommended, but possible).

The information contained in the descriptors is used by the disk device driver in level two OS9 to access the drives correctly, but unfortunately level one ignored the data and assumed that all drives were single-sided 35 track 30mS stepping. You can use DEBUG to examine the module in memory to see what your descriptor is set for. Here is a sample of how to use DEBUG to examine D0. Where you see this character sequence in my instructions <space> I mean for you to press the spacebar and this <ret> means press Enter.

```
OS9:debug
Interactive Debugger
DB: L<space>D0<ret>          ; this line links to the descriptor D0
   A000 87
DB: .<space>.+14             ; note the addresses and values may be different on your system
   A014 00
DB: .<space>.+2              ; this is the stepping rate 00 = 30mS
   A016 01
DB: <ret>                    ; this is the disk density 01 = double density
   A017 00
DB: <ret>                    ; this is the most significant byte of the number of tracks
   A018 23
DB: <ret>                    ; this is the least significant byte of the track size 23 = 35 tracks
   A019 01
DB: Q<ret>                   ; this is the number of sides. we use 'Q' to quit debug.
OS9:                         ; note that we did not modify anything, merely looked at the memory.
```

Please note here that I must make a confession. I ERRED IN THE DECEMBER ISSUE! The modpatch listing on page five should read as follows:-

```
modpatch
1 d0
c 14 00 03
c 16 01 03
c 18 23 50
c 19 01 02
v
```

I had got the address of the number of tracks right, but I modified the wrong byte (the MSB instead of the LSB) my apologies for misleading you. As a matter of fact modpatch would have reported the error by printing something like 'data does not match'.

The other device descriptor modules can be patched the same way, including the W, W1 ... W15 descriptors for level two windows. The only ones which do not need to be patched this way are P, the printer descriptor, and T1, T2, and T3, where XMODE is used to make the changes easier (and in english). For example, if you wanted to change the descriptor for W1 to different foreground, background and border colours, you could proceed this way:-

```
modpatch
1 w1
c 33 02 01
c 34 00 02
c 35 04 02
v
```

This will change the colours to blue, white and white for foreground, background, and border. To store this permanently, of course you must use os9gen or cobbler to save them to a new disk or the existing disk.
If you change a memory module using DEBUG, you must save the changed module to disk, and then use VERIFY with the U option to update the module's CRC. The easiest way is to use modpatch to do it for you and make sure to use the 'V' command after all the patches are done. This will force modpatch to reverify the module for you which means you can use cobbler to save the changes. If you do not update the CRCs and you cobbler to a new disk (or to the existing disk) you will not be able to reboot from it because of the incorrect CRCs.

Modifying the Printer speed.

Here we use the OS9 command XMODE. To change your printer baud rate, type the following:-

```
OS9:xmode /p baud=06
```

This changes the printer baud rate to 9600 Baud. You may find that it is also necessary to use the utility 'Tuneport' to get the printer working properly. The T1, T2, and T3 serial ports may be changed the same way. Note here that 'Tuneport' can also be used for T1, but not for T2 or T3.

Once you have changed your serial device descriptors (/p, /t1, /t2, /t3) you can cobbler a new disk and then these changes will be saved along with the rest of the modules in the OS9Boot file. You then won't have to wonder about whether you had the right baud rate etc, it is already done.

Well folks, that's all from me for this month,

Regards, Bob Devries.

oooooooooo0000000000oooooooooo

RUNNING A TERMINAL UNDER OS9 : FILE TRANSFERS

Last issue, we discussed how to set up a terminal on a CoCo OS9 system. Most of what I said then, applies to either Level I or Level II. The exception is the "immortal shell" command (ie Shell i=/T2&). This option is not available in the Level I kernel. OK, so now we have logged on to another machine, what can we do now?

Once you have logged onto a remote machine, and had a look around, I guess the most obvious thing to do is to try to transfer a file from that machine to your own. The normal way to do this would be to logoff the remote system, and get the operator of that system to manually transfer the file, using one of the common file transfer protocols. However some smart programmers in the USA have given us a better method. Enter the Kermit program. A great name for a programme that acts nothing at all like a frog!

This is the technique. Say, for example, that I am logged onto Bob Devries' CoCo, using Xcom9 as the terminal program on my own machine, and there is a file called newsletter_Jan.89 in the directory called TEXTS, in the root directory of his /D0 drive. I want to transfer a copy of that textfile to my own machine.

The basic technique is this:

1. Start up a sending kermit on his machine.
2. Quit the terminal program on my machine (but stay logged onto Bob's).
3. Start up a receiving kermit on my own machine to get the file.
4. Start up the terminal program again
5. Get back onto Bob's machine and say thanks.
6. Log off Bob's system.

Simple isn't it?

But in reality, it's not so simple.

Let's see if we can formulate the actual commands necessary to accomplish the file transfer. First of all, when the kermit program is run without any parameters, on an incorrect parameter, it provides the following help screen :

```
Kermit Program   Version 1    Release 6
Usage:   kermit c[le line esc.char]        (connect mode)
or:      kermit s[difl line] file ...      (send mode)
or:      kermit r[difl line]               (receive mode)
or:      kermit h[difl line]               (Host server mode)
or:      kermit g[difl line] file ...      (get file from server)
or:      kermit q[diful line]              (Quit remote Host server
```

What it does not provide however, is a description of what that all means, so I will add to it those definitions of which I am sure.

l ... add line feeds when outputting to screen

d ... one or more d's specifying debugging level. The more d's, the more verbose the output

i ... Specifies binary files. When this parameter is set, the file is transferred 'as is', when the parameter is not set, kermit assumes that the file is a text file, and translates <CR> to <CR><LF>

The remaining switches are to do with Connect and Host Server modes.

1. While we are logged onto Bob's machine, we start up a sending kermit. In order to do this the file that we are attempting to get must be in an accessible directory with its public-read attribute set, and the kermit program must have the public-execute attribute set (or we must be logged on as user 0, the superuser). OK, the command string then becomes :

kermit -sdi /d0/TEXTS/newsletter_Jan.89

Remember, we are logged onto his machine through /T2, and kermit is smart enough to know that, so we don't have to specify the line we are using. In fact if we do, kermit will think that /T2 is a file name, and try to look for it to send. Assuming everything goes to plan, that command should start a sending kermit on his machine, which will sit and wait for an ACK signal on the line.

2. We then get out of the terminal program on our own machine (when using Xcom9 <CTRL>+Y, Q).

3. Now using our own system, with kermit in our own execution directory, we start a receive kermit by typing:

kermit -rdi /T2

The kermit protocol transfers filenames (but not full pathnames) so that you will not have to specify a filename when starting your receive kermit. A word of warning though, if you happen to have a file of the same name in your current data directory, it will be overwritten, and there will be no error reported!

After the process has terminated, there will now be a file called newsletter_Jan.89 in our current data directory.

4. Start our terminal program again in the usual manner, and a <CR> should give us a shell prompt from Bob's system (because we are still logged on). We could then relay a message to him to tell him that we are finished etc.

A nifty technique that we sometimes use to do this involves redirection of a message to a window without a shell running. If that window happens to be W2, I would use the following:

```
echo Thanks Bob, go to voice >/W2     *** This is the message
display 07 07 07 >/W2                 *** This gets his attention
```

Remember, it is important that the window used has no shell, as the message will then be displayed immediately without having to wait for it to become active.

Both kermit, and Xcom9 are public domain programs, and are available from us. Simply send us a formatted disk, sufficient stamps to cover return postage, and $2.00, and we will provide copies. There are a number of versions of both kermit and Xcom9. The earlier releases of kermit will not work satisfactorily on a CoCo, I think because they use a system call which is not available on CoCo OS9 Level I. I use Version 1, release 6. There are differing versions of Xcom9 available dependent on OS9 Level, so please specify which Level you require.

Any questions should be directed to me (Don Berrie), on (07) 375 3236, or Bob Devries on (07)372 7816.

oooooooooo0000000000oooooooooo

FOR SALE :- OS9 Software Package.
Software by TANDY to suit CoCo3, OS9 Level II complete with Basic09, Pascal language compiler, 'C' language compiler, Dynacalc spreadsheet, Profile database, Rainbow "OS9 Level II Book" - ALL original manuals and disks in as new condition. Cost over $800 BARGAIN at $250 the lot. Phone (07) 349 4696 ask for Brian.

oooooooooo0000000000oooooooooo

A Celcius to Fahrenheit Convertor in 'C'

Here's a little programme that will win you much esteem with your wife. It is a programme to give a list of conversions for temperatures converting from degrees F to degrees C and back again. It has the ability to be printed to your printer if you so desire. The programme asks you if you wish to convert C to F or F to C just press C or F to select. You will then be asked what temperature to start at, then what to stop at and what step value to use, just like a FOR-NEXT loop (which it is used for in fact). You will then be presented with a list of temperatures and their conversions. Programme output is as follows :-

```
Convert C to F press C
   or   F to C press F
        your choice ? C

Value to start at ? 100
Value to end at ? 200
Step value ?  10

 100 degrees C =  212 degrees F
 110 degrees C =  230 degrees F
 120 degrees C =  248 degrees F
 130 degrees C =  266 degrees F
 140 degrees C =  284 degrees F
 150 degrees C =  302 degrees F
 160 degrees C =  320 degrees F
 170 degrees C =  338 degrees F
 180 degrees C =  356 degrees F
 190 degrees C =  374 degrees F
 200 degrees C =  392 degrees F
```

As you can see I allowed only whole numbers to be printed not the fractional values. This can be changed in line 41 of the programme by changing %4.0f to %4.2f to change it to 2 decimal places. To get the programme to print to the printer, you need to redirect standard error. Use the following command line:-

```
OS9:tempconv >>/p
```

This will only print out the conversion lines not the instructions. It is very useful for producing a list of oven temperatures for the kitchen. Now without further ado, here's the listing....

```
#include <stdio.h> /* for the getchar() macro */

main()
{
        float count;/* for-next loop counter variable */
        float first;/* starting value for loop        */
        float last;/* ending value for loop          */
        float step;/* step value for loop            */
        float result;/* result returned from conversion*/
        char ch;/* keyboard input variable        */
        float F_to_C();/* declare function as returning  */
        float C_to_F();/* a float value                  */
        pffinit();/* tell linker we are using float */

        printf("Convert C to F press C\n");/* print instructions */
        printf("   or   F to C press F\n");
        printf("        your choice ? ");
```

```
        ch = toupper(getchar());/* get a key from user */
        if (ch != 'F' && ch != 'C')/* if not C or F end programme */
                exit(0);
        printf("\n\nValue to start at ? ");/* get start value */
        scanf("%f",&first);
        printf("Value to end at ? ");/* get ending value */
        scanf("%f",&last);
        printf("Step value ? ");/* get step value */
        scanf("%f",&step);

        for (count=first;count<=last;count+=step)/* for-next loop */
                {
                switch (ch)/* use switch to select conversion type */
                        {
                        case 'F':/* if F convert F to C */
                                result = F_to_C(count);
                                break;
                        case 'C':/* if C convert C to F */
                                result = C_to_F(count);
                                break;
                        }
/* now print the result to standard error output to permit re-direction */
                fprintf(stderr,"%4.0f degrees %c = %4.0f degrees %c\n",count,ch,result,(ch == 'F' ? 'C' : 'F'));
                }
}

float F_to_C(degrees)/* fahrenheit to celcius conversion function */
float degrees;/* value passed to routine is here */
{
        return ((5.0 / 9.0) * (degrees - 32.0));/* do your sums */
}

float C_to_F(degrees)/* celcius to fahrenheit conversion function */
float degrees;/* value passed to routine is here */
{

        return (((9.0 / 5.0) * degrees) +32.0);/* do your sums */
}
```

I hope you find this programme useful as well as a little educational. Have fun with it anyhow.

Regards,
Rob Devries

oooooooooo0000000000oooooooooo